Decidability in Parameterized Verification*

Roderick Bloem[1] Swen Jacobs[2] Ayrat Khalimov[1] Igor Konnov[3]
Sasha Rubin[4] Helmut Veith[3] Josef Widder[3]
[1] TU Graz
[2] Universität des Saarlandes
[3] TU Wien
[4] Università degli Studi di Napoli "Federico II"

Abstract

Parameterized model checking is an active research field. The system models that it considers, as well as the proof methods it uses for obtaining decidability results, are quite similar to those considered in distributed computing. We hope that our recent book [11] helps distributed computing experts in understanding and entering parameterized model checking research. In this short note we want to give a taste of this area.

1 Introduction

Designing concurrent or distributed systems, and proving their correctness, is both difficult and error-prone. There are, roughly speaking, two streams of research that address these tasks. On the one hand, principles of distributed computing are studied in order to develop mathematical proof methods that allow to establish complexity results in distributed computing (lower and upper bounds) [34, 8]. On the other hand, one develops computer aided verification methods, such as model checking [14]. Mathematical proofs typically consider parameterized systems, where, e.g., the parameter n determines the number of replicas in a distributed system, and one determines the correctness of a distributed algorithm for all values of n. The classic, and largely outdated, definition of model checking is that it considers fixed size and finite state systems only. That is, one fixes e.g. $n = 4$ and checks the small system for presence of bugs. However, nowadays most research in model checking considers parameterization in one way or another, e.g., parameterized number of processes, or parameterized domain of variables. Consequently, there is a vast literature in this domain.

Our recent book [11] focuses on automatic verification of systems where the number of processes is parameterized, while the local state space of the processes is finite. More precisely, we consider decidability of parameterized model checking of concurrent systems. The literature in this field considers different computational models and we thus survey results for the most studied models: token passing systems, systems where processes coordinate by

*Because of his tragic death, Helmut Veith did not participate in finishing this note. His curiosity and energy ignited our joint long-term effort in parameterized verification.

This work was supported by the Austrian National Research Network RiSE (S11403, S11405, S11406) and project PRAVDA (P27722) of the Austrian Science Fund (FWF), by the Vienna Science and Technology Fund (WWTF) through grants APALACHE (ICT15-103) and PROSEED, by the German Research Foundation (DFG) through SFB/TR 14 AVACS and project ASDPS (JA 2357/2-1), and by the Istituto Nazionale di Alta Matematica through INdAM-COFUND-2012, FP7-PEOPLE-2012-COFUND (Proj. ID 600198).

pairwise rendezvous or one-to-many synchronization (broadcast), guarded protocols, and finally ad-hoc networks.

In this note, we briefly discuss the features of these computational models and some conclusions we have drawn from surveying the literature. For formal definitions as well as a detailed survey of the results and their proofs we refer to the book. We start with discussing the computational model used in the book in Section 2 and some often-used proof techniques in Section 3, and then describe the different system models that are covered in the book, and summarize their decidability results in Sections 4 to 7.

2 Models and Specifications

Finding out whether parameterized model checking is decidable for one's favorite computational model boils down to checking whether it is captured by the semantics of one of the published computational models. However, computational models are scattered over the literature and use different terminology, and results are based on slightly different assumptions. Hence, it is cumbersome to check whether existing results apply to our favorite models. Our motivation was to provide a one-stop-source for results that eases this task. This required us to provide a definition for concurrent systems that incorporates many of the foundational computational models in the parameterized model checking literature. The models we discuss in Sections 4 to 7 specialize specific features of this general model.

In our framework, a concurrent system, or *system instance*, $\overline{P}^G$ is composed of a vector of *process templates* $\overline{P} = (P^1, \ldots, P^{\mathsf{d}})$, and a graph G, on which copies of the templates are arranged. We consider discrete time, and at each time step either one process acts alone, or some process v initiates an action and some set of processes that are connected to v in G simultaneously synchronize with v. Thus, we consider that processes coordinate instantaneously using synchronization primitives (e.g., rendezvous), and do not consider, e.g., non-blocking communication by message buffers. Most importantly, the process templates we consider have a fixed and finite local state space.

Synchronization primitives can then be defined by restricting the number of processes that can simultaneously synchronize with the initiating process. We do so by a so-called *synchronization constraint* that specifies how many processes should synchronize. For instance, pairwise rendezvous is captured by having the singleton $\{1\}$ as synchronization constraint, which means that every synchronous transition must be taken by the initiating process and exactly one other process.

Then, a *parameterized system* is a sequence of system instances formed from a fixed vector of process templates and a sequence of graphs $\mathbf{G}$. Typical sequences of graphs are rings of size n, cliques of size n, or stars of size n, where n is the parameter. The nth instance of a parameterized system is the system instance $\overline{P}^{\mathbf{G}(n)}$. *Parameterized specifications* make statements about parameterized systems by quantifying over process indices (i.e., vertices of $\mathbf{G}(n)$), e.g., "every process v of $\mathbf{G}(n)$ eventually satisfies predicate p". To formalize this, we recall the definitions of several *indexed temporal logics* in the book. The *parameterized model checking problem (PMCP)* is to decide whether for all $n \in \mathbb{N}$ the instance $\overline{P}^{\mathbf{G}(n)}$ satisfies the specification.

3 Proof Techniques

3.1 Undecidability

In this area, undecidability is typically proven by reduction from the non-halting problem of two-counter machines, which is known to be undecidable. In the proof, one then shows how to simulate up to n steps of a two-counter machine in a system composed of $n + 1$ finite-state processes: There is a *controller* process that simulates the internal state of the two-counter machine. The remaining n processes collectively encode the counter values in a way that allows for storing values up to n, which is the maximal value that can be written in n steps. The proofs differ in how the controller uses the synchronization primitives to issue commands to increment/decrement counters and test counters for zero.

The prototypical such proof in a token ring is by Emerson and Namjoshi [27] and describes a distributed algorithm that performs this task. This improved the first undecidability results for systems consisting of identical processes arranged in a uni-directional ring with a single multi-valued token by Suzuki [41] who reduces non-halting of Turing machines to parameterized model checking. Undecidability of parameterized model checking then follows from undecidability of the non-halting problem.

3.2 Decidability

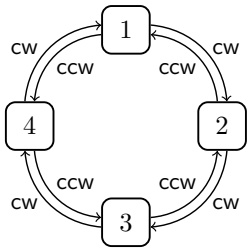
One way to prove decidability of parameterized verification for a certain class of concurrent systems is to show that this class can be represented as a well-structured transition system [29]. For such systems Abdulla et al. [2] provided a comprehensive theory.

A quite intuitive approach to parameterized model checking is to check the system for small values of n and *assume* that if there is a bug in a system with a large number of components, then the bug already appears in the small system. This approach can be formalized as a *cutoff* or *decomposition* statement [27, 15] that reduces the parameterized model checking problem to a finite collection of classic model checking problems. For specific classes of systems one can show that such cutoffs or decompositions indeed exist, from which decidability of parameterized model checking for such systems follows. To prove that c is a cutoff for a class of systems and specifications in an indexed temporal logic, it suffices to show that a system with exactly c components relates to a system with arbitrarily many components. The details of the specific relation (e.g., simulation or bisimulation) depend on the class of systems and the temporal logic under consideration.

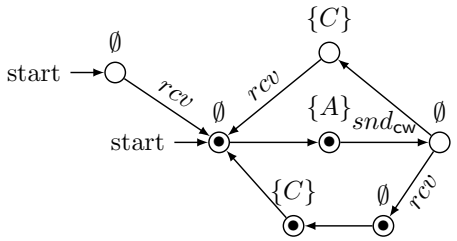
4 Token-passing Systems

In a token-passing system (TPS), processes communicate by passing a token in the network. The token may or may not carry a value that can be updated by the process that holds it. In case of a valueless token, the only purpose of the token is to distinguish the process that currently holds it from all the other processes, which allows one for example to implement systems with mutual exclusion properties.

A well-known example system that can be modeled as a TPS with valueless token is *Milner's scheduler* [36]. The scheduler is composed of an arbitrary number of components, each of which is responsible for activating some (unspecified) task and receives confirmation when its task has been executed. Scheduling should ensure that the tasks are activated in a round-robin scheme, and that every task has terminated before being activated again. As noted by Emerson and Namjoshi [27], this can naturally be modeled in a token ring. That



(a) A bi-directional ring with 4 nodes. Edges are labeled with directions cw (clockwise) or ccw (counterclockwise). Processes can send the token in a direction by using actions snd_{cw} or snd_{ccw}, respectively.



(b) Process template for Milner's scheduler. States with token are depicted with a dot inside. State labels are depicted above states. Label A stands for activation of the task, C for completion of the task.

Figure 1: Milner's scheduler in a token ring.

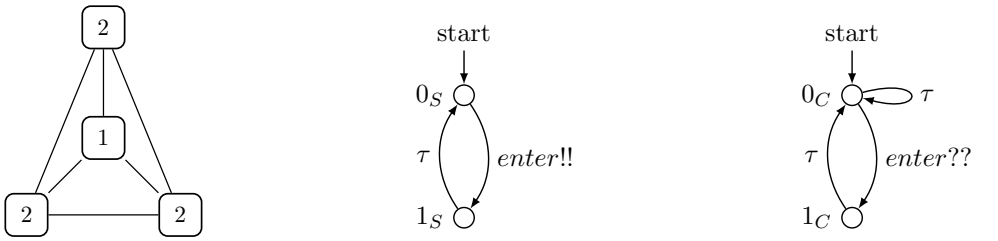
is, processes are arranged in a ring, and one process starts with the token. A process that has the token will activate its task, and then send the token to the next process. After starting the task, it will wait for both the return of the token and a confirmation that the task has terminated. Only after both of these events, the process will activate its task again. Thus, communication by a single valueless token is sufficient to guarantee the global properties required by the Milner scheduler. The graph and process template for Milner's scheduler in a token ring are depicted in Figure 1.

For other classes of systems, the parameterized verification literature usually considers a single fixed graph structure, like cliques for systems with broadcast communication or guarded protocols. In contrast, TPSs have been analyzed on a variety of graphs that can be much more complex. In particular, this includes graphs where connections may be labeled with directions, and processes can choose into which direction they want to send the token. Finally, another orthogonal extension of the model considers systems with multiple tokens.

Results. Most of the decidability results in the literature are for TPSs with a single valueless token [27, 15, 5], and where token passing satisfies some notion of fairness. However, even in systems with such a restricted communication primitive, parameterized model checking is undecidable if we consider arbitrarily complex graphs and specifications in a branching-time logic. Undecidability proofs show that such systems can simulate two-counter machines, and deciding the PMCP would decide the non-halting problem of these machines [15, 5]. To obtain decidability results, the literature considers restrictions on the graph structure, as well as on the number of index and path quantifiers in the specification.

For rings, Emerson and Namjoshi [27] have identified concrete cutoffs between 2 and 5, depending on the specification, but independent of the process template. Cutoffs for rings and some other classes of graphs can be found by manually constructing a bi-directional simulation as mentioned in Section 3.2 [27, 5]. In TPSs with possibly complex graphs, this simulation needs to take into account whether processes mentioned in the specification are direct neighbors or not, and consider all possible cases of neighborship relations if the specification quantifies over multiple processes. For certain classes of graphs that can be constructively specified, cutoffs can also be computed automatically [6].

If tokens can carry values, decidability is lost even in uni-directional rings [41]. Similarly, if processes can distinguish directions in the graph, we get undecidability in bi-directional rings [5]. To recover decidability, additional restrictions on graphs, processes, or specifications are necessary.



(a) A clique graph with three clients and one server.

(b) A server process.

(c) A client process.

Figure 2: A Client/Server system.

5 Broadcast and Asynchronous/Pairwise-Rendezvous

The broadcast communication primitive is an abstraction of, e.g., ethernet-like broadcast, GSM's cell-broadcast, communication in Prasad's Calculus of Broadcasting Processes (CBP) [39], or the notifyAll method in Concurrent Java [18]. To use the broadcast communication primitive, an initiator process sends a message and every process that is able to receive the message (if any) immediately and simultaneously does so. Figure 2 illustrates a client-server system with three clients. If the server broadcasts the message *enter* by firing the transition from state 0_S to 1_S, then, simultaneously, every client that is in state 0_C transitions to 1_C.

Two other related primitives are asynchronous-rendezvous in which exactly one process that is able to receive the message does so (if any), and pairwise-rendezvous which is like asynchronous-rendezvous except that sending is blocked if there is no process that is able to receive. Asynchronous-rendezvous is an abstraction of, e.g., the notify method in Concurrent Java [18], and pairwise-rendezvous is like synchronized communication in Milner's Calculus of Communicating Systems (CCS) and Hoare's Communicating Sequential Processes (CSP).

Broadcast can express many others primitives including asynchronous- and pairwise-rendezvous, token-passing (Section 4) and disjunctive guards (Section 6) [7].

Results. We assume the communication graph is a clique (other graphs are assumed in Sections 4 and 7). Parameterized model checking of broadcast systems is undecidable for liveness properties (e.g., some state from a given set is seen infinitely often), and decidable for safety properties (e.g., no state from a given set is ever visited). The proof of undecidability uses the basic encoding of two counter machines outlined above. The interesting part of the simulation is testing a counter for zero, and uses a trick in which the controller broadcasts a guess of whether the counter is zero or not, thus potentially introducing errors to the simulation, which can be compensated for, however. Decidability of safety properties follows from the fact that broadcast systems are well-structured transition systems [2]. The case of asynchronous-rendezvous is similar.

On the other hand, for pairwise-rendezvous, both liveness and safety specifications are decidable. The proof of this fact shows that such systems can be expressed as Petri Nets or Vector Addition Systems [28]. The main idea is to use a counter-representation that captures, for each configuration, the number of processes in a each state. Thus configurations are represented as vectors of natural numbers, and a rendezvous between two processes corresponds to adding a fixed vector to a configuration.

All the results above are for linear-temporal specifications. Branching-temporal specifications are undecidable already for pairwise rendezvous [6].

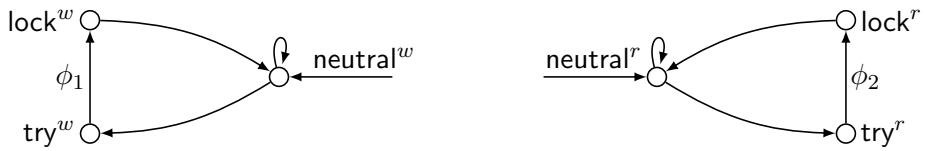


Figure 3: A readers/writer protocol.

6 Guarded Protocols

The key feature of guarded protocols is that every process can query the local states of the other processes in an anonymous way. For instance, a process j evaluates the disjunctive guard $[\exists\ \mathsf{other}\ i]\ A_i \vee B_i$ to true, whenever a global state contains a process i (other than j) that is either in local state A, or B. Likewise, a process j evaluates the conjunctive guard $[\forall\ \mathsf{other}\ i]\ A_i \vee B_i$ to true, whenever all processes different from j are either in A, or in B. While the seminal paper by Emerson and Namjoshi [26] introduced guarded protocols for synchronous systems, the follow-up papers by Emerson and Kahlon [23, 24, 25] considered asynchronous guarded protocols.

Guarded protocols have been used to model cache-coherence protocols. A more classical example is a *multiple* readers and *single* writer protocol shown in Figure 3. The readers start in the state $\mathsf{neutral}^r$, and the writers start in the state $\mathsf{neutral}^w$. A writer's transition from the trying state try^w to the locking state lock^w is guarded with the guard $\phi_1 \equiv [\forall\ \mathsf{other}\ j]\ \mathsf{neutral}^r_j \vee \mathsf{neutral}^w_j \vee \mathsf{try}^r_j \vee \mathsf{try}^w_j$, which forbids this writer to make a transition, if some readers or writers are in the locking state. A reader makes a transition from the trying state try^r to the locking state lock^r, *only* if all writers are in the neutral state, that is, the guard ϕ_2 is $[\forall\ \mathsf{other}\ j]\ \mathsf{neutral}^r_j \vee \mathsf{neutral}^w_j \vee \mathsf{try}^r_j \vee \mathsf{lock}^r_j$.

Results. Unfortunately, parameterized model checking of boolean guarded protocols — that is, protocols that use both disjunctive and conjunctive guards — is undecidable [26]. As usual, undecidability is shown by simulating two-counter machines. Thus, the research has been focused on verification of systems that have only one kind of guards [23, 24, 25].

As shown by Emerson and Kahlon [23], each system composed of n processes having Q local states and using only disjunctive guards has a cutoff of size $|Q| + 2$. Hence, to check a temporal property for all system sizes, it is sufficient to check the property for systems of size up to $|Q| + 2$. A cutoff of size $2|Q| + 1$ was also found [23] for systems with conjunctive guards having the following restrictions: every process can remain indefinitely long in the initial state, and no guard evaluates to false due to a process staying in the initial state.

Further, Emerson and Kahlon showed that systems with disjunctive guards can be simulated by systems with rendezvous communication [25]. Hence, one achieves decidability of PMCP for the systems that use both disjunctive guards and rendezvous communication.

In most cases, extending guarded protocols with broadcast leads to undecidability [25]. The only known decidability result is for regular action-based properties [25], that is, the properties that can be described with a finite automaton recognizing words over the alphabet of action labels.

Finally, Emerson and Kahlon developed special techniques for the restricted versions of guarded protocols that model cache coherence protocols [24].

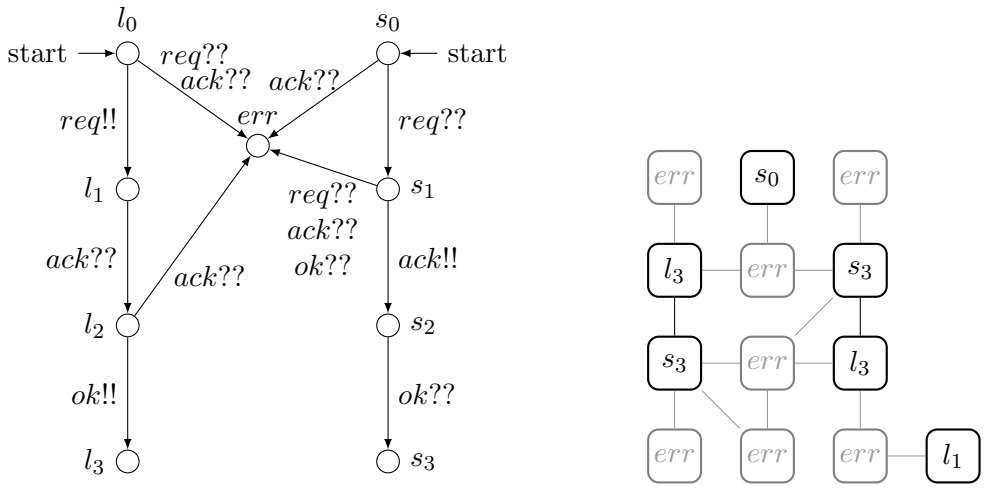


Figure 4: On the left: Request-Acknowledgment-OK protocol for ad hoc networks [19]: initial states are $\{s_0, l_0\}$, $a!!$ means "broadcast message a", and $a??$ means "receive message a" (for $a \in \{req, ack, ok\}$). On the right: example state after running the protocol.

7 Ad Hoc Networks

In ad hoc networks, processes communicate using "local" broadcasts: such messages are visible only to processes in the communication range of the sender. Ad hoc networks can be captured by broadcast systems (Section 5) if we model the communication ranges using the system graph: two nodes are connected only if in the ad hoc network the two processes can hear each other.

Figure 4 illustrates a Request-Acknowledgment-OK protocol to be run by processes of an ad hoc network, and a possible state after executing the protocol. The intention of the protocol is to "create" pairs of connected processes in states l_3 and s_3 that are isolated from others by processes in state err.

In contrast to other systems considered in our survey, the literature on ad hoc networks [19, 20, 21, 1] does not study the PMCP for a general class of specifications, but rather three specific problems called COVER, REPEAT, and TARGET. COVER asks, given a process description and a process control state, whether the control state can be reached in a network of some size. Similarly, REPEAT asks if the state can be reached infinitely often, and TARGET asks if all processes can simultaneously reach the state. The first two problems can be expressed as a PMCP using the prenex indexed temporal logic, but the last one requires a non-prenex fragment.

The literature also differentiates the results for different classes of graphs. We survey results for five classes of undirected graphs, inspired by those found in practice, including cliques, wheels, stars, processes arranged in clusters, and unrestricted graphs. We also survey results [1] for directed graphs. Such graphs can model discrepancies in the send and receive transmission ranges of processes.

Finally, the literature distinguishes ad hoc networks with and without failures. The failure model is: any broadcast message can be lost non-deterministically for some of the recipients. A related model is that of mobile networks, in which the network graph can non-deterministically change during the execution. Delzanno et al.[21] showed that the models of lossy and mobile networks are equivalent, so we focus on the former only.

Results. Roughly, COVER for ad hoc networks without failures is undecidable on graphs that can have simple paths of unbounded length, except cliques (e.g., wheel-like graphs) [19]. The undecidability proof uses the ability of the network to simulate a given two-counter machine, as follows. First, the network runs a special protocol that shuts down all processes except one leader connected to two lists of processes. Then, the leader simulates the control of the two-counter machine using the two lists to store the values of the counters. In this construction, to be able to store the unbounded values of the counters, the two lists should be of the unbounded length (unbounded in the parameterized system).

COVER is decidable on graphs in which all simple paths have a bounded length [19, 20]. The decidability proofs go via the machinery of well-structured transitions systems. Intuitively, the proofs use the insight that behaviours of an infinite family of networks have a finite basis, and it can be found using a saturation algorithm. Note that REPEAT and TARGET are undecidable on all graph classes we survey, if considered on ad hoc networks without failures [19].

All three problems become decidable for the case of lossy ad hoc networks [21]. The hostile failure model makes it impossible to simulate a two-counter machine, and the decidability proofs go via reduction to Petri nets (a certain kind of well-structured transition systems).

8 Conclusions

We believe that parameterized model checking and distributed computing could mutually benefit from studying parameterized verification in the context of state-of-the-art computation models for distributed algorithms. To initiate a dialog between the concerned communities, in this short note we wanted to give the distributed computing community a taste of the questions studied in parameterized model checking. More details can be found in our recent book [11].

While our initial goal in writing the book was to give a comprehensive survey on parameterized model checking, we quickly learned that the field is too lively for that. Consequently, we limited ourselves to decidability issues in concurrent systems of identical finite state processes. Indeed, our survey is just one in the area of parameterized model checking. Most recently, Delzanno [17] surveyed broadcast protocols. Before that Zuck and Pnueli [43] surveyed abstraction techniques. Regular model checking was surveyed by Abdulla [4], well-structured transition systems were surveyed by Finkel and Schnoebelen [29], and decidability in Petri nets was surveyed by Esparza [28].

Several parameterized model checking techniques have been implemented in tools such as BOOM [10], ByMC [12], T(O)RMC [33], TLV [38] and Jtlv [37], Undip [3], MCMT [30], CheAPS [32], MCMAS-P [35], and Cubicle [16]. Furthermore, there is a number of tools that implement more general forms of infinite-state model checking, in particular with support for integer-valued variables. With a suitable encoding, these tools can be used for parameterized model checking. Examples of such tools are ALV [42], BRAIN [40], FAST [9], MONA [22], and nuXmv [13]. Finally, the tool Party [31] uses decidability results surveyed in our book.

Acknowledgments. We are grateful to Paul Attie, Giorgio Delzanno, Sayan Mitra, and Kedar Namjoshi for carefully reading a draft of our book, and providing detailed and constructive comments.

References

[1] P. A. Abdulla, M. F. Atig, and O. Rezine. Verification of directed acyclic ad hoc networks. In *FORTE*, volume 7892 of *LNCS*, pages 193–208. Springer, 2013.

[2] P. A. Abdulla, K. Čerāns, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *LICS*, pages 313 –321, 1996.

[3] P. A. Abdulla, G. Delzanno, and A. Rezine. Approximated parameterized verification of infinite-state processes with global conditions. *Formal Methods in System Design*, 34(2):126–156, 2009.

[4] P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. A survey of regular model checking. In *CONCUR*, volume 3170 of *LNCS*, pages 35–48. Springer, 2004.

[5] B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *VMCAI*, volume 8318 of *LNCS*, pages 262–281, Jan. 2014.

[6] B. Aminof, T. Kotek, S. Rubin, F. Spegni, and H. Veith. Parameterized model checking of rendezvous systems. In *CONCUR*, volume 8704, pages 109–124. Springer, 2014.

[7] B. Aminof, S. Rubin, and F. Zuleger. On the expressive power of communication primitives in parameterised systems. In M. Davis, A. Voronkov, A. McIver, and A. Fehnker, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, 2015.

[8] H. Attiya and J. Welch. *Distributed Computing*. John Wiley & Sons, 2nd edition, 2004.

[9] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. FAST: acceleration from theory to practice. *STTT*, 10(5):401–424, 2008.

[10] G. Basler, M. Mazzucchi, T. Wahl, and D. Kroening. Symbolic counter abstraction for concurrent software. In *CAV*, volume 5643 of *LNCS*, pages 64–78. Springer, 2009.

[11] R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.

[12] ByMC. ByMC: Byzantine model checker, 2013. URL: `http://forsyte.tuwien.ac.at/software/bymc/`. Accessed: April, 2016.

[13] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta. The nuXmv symbolic model checker. In *CAV*, volume 8559 of *LNCS*, pages 334–342, 2014.

[14] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[15] E. Clarke, M. Talupur, T. Touili, and H. Veith. Verification by network decomposition. In *CONCUR 2004*, volume 3170, pages 276–291, 2004.

[16] S. Conchon, A. Goel, S. Krstic, A. Mebsout, and F. Zaïdi. Cubicle: A parallel smt-based model checker for parameterized systems - tool paper. In *CAV*, volume 7358 of *LNCS*, pages 718–724. Springer, 2012.

[17] G. Delzanno. A unified view of parameterized verification of abstract models of broadcast communication. *International Journal on Software Tools for Technology Transfer*, pages 1–19, 2016.

[18] G. Delzanno, J. Raskin, and L. Van Begin. Towards the automated verification of multithreaded Java programs. In *TACAS*, volume 2280 of *LNCS*, pages 173–187, 2002.

[19] G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR*, volume 6269 of *LNCS*, pages 313–327, 2010.

[20] G. Delzanno, A. Sangnier, and G. Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *FOSSACS*, volume 6604 of *LNCS*, pages 441–455. Springer, 2011.

[21] G. Delzanno, A. Sangnier, and G. Zavattaro. Verification of ad hoc networks with node and communication failures. In *FORTE*, volume 7273 of *LNCS*, pages 235–250. Springer, 2012.

[22] J. Elgaard, N. Klarlund, and A. Møller. MONA 1.x: new techniques for WS1S and WS2S. In *CAV*, volume 1427 of *LNCS*, pages 516–520. Springer, 1998.

[23] E. A. Emerson and V. Kahlon. Reducing model checking of the many to the few. In *CADE*, volume 1831 of *LNCS*, pages 236–254. Springer Berlin Heidelberg, 2000.

[24] E. A. Emerson and V. Kahlon. Exact and efficient verification of parameterized cache coherence protocols. In *CHARME*, volume 2860 of *LNCS*, pages 247–262. Springer, 2003.

[25] E. A. Emerson and V. Kahlon. Model checking guarded protocols. In *LICS*, pages 361–370. IEEE, 2003.

[26] E. A. Emerson and K. S. Namjoshi. Automatic verification of parameterized synchronous systems. In *CAV*, volume 1102 of *LNCS*, pages 87–98. Springer, 1996.

[27] E. A. Emerson and K. S. Namjoshi. On reasoning about rings. *Int. J. Found. Comput. Sci.*, 14(4):527–550, 2003.

[28] J. Esparza. Decidability and complexity of petri net problems - an introduction. In *In Lectures on Petri Nets I: Basic Models*, pages 374–428. Springer-Verlag, 1998.

[29] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.

[30] S. Ghilardi and S. Ranise. Backward reachability of array-based systems by SMT solving: Termination and invariant synthesis. *Logical Methods in Computer Science*, 6(4), 2010.

[31] A. Khalimov, S. Jacobs, and R. Bloem. PARTY parameterized synthesis of token rings. In *CAV*, volume 8044 of *LNCS*, pages 928–933. Springer, 2013.

[32] I. V. Konnov and V. A. Zakharov. An invariant-based approach to the verification of asynchronous parameterized networks. *J. Symb. Comput.*, 45(11):1144–1162, 2010.

[33] A. Legay. T(O)RMC: A tool for (omega)-regular model checking. In *CAV*, volume 5123 of *LNCS*, pages 548–551. Springer, 2008.

[34] N. Lynch. *Distributed Algorithms*. Morgan Kaufman Publishers, Inc., San Francisco, USA, 1996.

[35] MCMAS-P. VAS – Verification of autonomous systems, 2016. URL: `http://vas.doc.ic.ac.uk/software/extensions/`. Accessed: April 2016.

[36] R. Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.

[37] A. Pnueli, Y. Sa'ar, and L. D. Zuck. JTLV: A framework for developing verification algorithms. In *CAV*, volume 6174 of *LNCS*, pages 171–174. Springer, 2010.

[38] A. Pnueli and E. Shahar. A platform for combining deductive with algorithmic verification. In *CAV*, volume 1102 of *LNCS*, pages 184–195. Springer, 1996.

[39] K. V. S. Prasad. A calculus of broadcasting systems. *Sci. Comput. Program.*, 25(2-3):285–327, 1995.

[40] T. Rybina and A. Voronkov. BRAIN : Backward reachability analysis with integers. In *AMAST*, volume 2422 of *LNCS*, pages 489–494. Springer, 2002.

[41] I. Suzuki. Proving properties of a ring of finite-state machines. *Inf. Process. Lett.*, 28(4):213–214, July 1988.

[42] T. Yavuz-Kahveci and T. Bultan. Action language verifier: an infinite-state model checker for reactive software specifications. *Formal Methods in System Design*, 35(3):325–367, 2009.

[43] L. D. Zuck and A. Pnueli. Model checking and abstraction to the aid of parameterized systems (a survey). *Computer Languages, Systems & Structures*, 30(3-4):139–169, 2004.